Bugiardino tecnico

OII Staff

3 novembre 2018

Indice

1 Shell **2**

2 Compilatore **2**

3 Debugger **3**

4 STL **4**

1 Shell

Esempi di comandi interessanti sono:

`man echo`	Visualizza il manuale del comando specificato.
`echo *.cpp ??? '*'`	Stampa tutti i nomi di file nella cartella corrente che terminano con ".cpp", tutti quelli il cui nome è di tre caratteri e un asterisco.
`pwd`	Stampa la cartella in cui ti trovi.
`cd cartella`	Entra nella cartella "cartella".
`cd ..`	Sale di un livello nella gerarchia delle cartelle.
`ls`	Stampa l'elenco dei file contenuti nella cartella corrente.
`mv`	Rinomina o sposta un file.
`cp`	Copia uno o più file.
`rm`	Cancella uno o più file.
`wc output.txt`	Conta il numero di caratteri, parole e righe di ciò che riceve in input.
`cat input.txt`	Stampa il contenuto di un file di testo.
`less programma.cpp`	Mostra il contenuto di un file di testo in maniera interattiva.
`sort -n`	Ordina numericamente le righe di ciò che riceve in input.
`grep "ciao"`	Filtra le righe contenenti l'espressione regolare specificata ("ciao")
`sed "s/int/long/g"`	Sostituisce l'espressione regolare specificata.
`./programma`	Esegue il programma che è presente nella cartella corrente.
`cat output.txt | sort`	Redirige l'output del primo comando come input per il secondo.
`echo 123 >  input.txt`	Memorizza l'output del primo comando su un file esterno.
`./programma < input.txt`	Fornisce al comando il file specificato come input.
`i=17355`	Assegna un valore a una variabile.
`echo $[i*i+32]`	Stampa il risultato dell'operazione numerica.
`echo ${#i} ${i:1:3}`	Stampa la lunghezza della stringa e la sottostringa indicata.

`for ((i=0; i<10; i++));`
`  do echo '$i =' "$i";`
`done`
	Stampa 10 righe numerate.

`for i in *.pas;`
`  do mv "$i" $(echo $i \| sed "s/pas$/cpp$/g");`
`done`
	Rinomina tutti i file `.pas` in `.cpp`.

2 Compilatore

Per compilare un programma, utilizzate:

```
g++ [opzioni] programma.cpp -o programma
```

Per compilare con un grader, utilizzate:

```
g++ [opzioni] grader.cpp programma.cpp -o programma
```

Opzioni interessanti sono:

`-g`	Genera simboli di debug (non usare con `-O2`).
`-O2`	Il programma viene ottimizzato al livello 2.
`-std=c++11`	Il programma viene compilato secondo lo standard del C++ 11.
`-Wall`	Mostra molti warning.
`-Wextra`	Mostra ancora più warning.
`-pedantic`	Mostra più warning che può.
`-ftrapv`	Il programma viene compilato per controllare gli integer overflow.
`-fstack-protector-all`	Il programma viene compilato per controllare i buffer overflow.
`-fsanitize=address`	Il programma viene compilato per controllare le scritture fuori memoria.
`-D_GLIBCXX_DEBUG`	Richiede alla libreria standard di controllare gli errori più comuni.

3 Debugger

`gdb programma`	Apre il debugger da linea di comando con il programma specificato (compilato con `-g`).
`ddd programma`	Apre il debugger grafico con il programma specificato (compilato con `-g`).

Comandi per il debugger interessanti sono:

`run`	Esegue il programma dall'inizio fino al primo breakpoint.
`start`	Esegue il programma dall'inizio fino all'apertura del main.
`step`	Esegue una nuova riga di codice, entrando nelle chiamate a funzione.
`next`	Esegue una nuova riga di codice, senza entrare nelle chiamate a funzione.
`continue`	Prosegue l'esecuzione del programma fino al successivo breakpoint.
`backtrace`	Stampa la lista delle chiamate che hanno portato al passo di esecuzione corrente.
`break`	Imposta o disabilita un breakpoint.
`print`	Stampa il valore di una variabile.
`display`	D'ora in poi mostra il valore di una variabile.
`watch`	L'esecuzione verrà interrotta appena la variabile specificata cambierà di valore.
`<invio>`	Esegue nuovamente l'ultimo comando inserito.

4 STL

Inserite all'inizio del vostro programma la seguente riga:

```
using namespace std;
```

Oggetti interessanti sono:

Oggetto	include	descrizione
`pair<T1, T2>`	`utility`	Coppia che si crea con `make_pair(a, b)`.
`vector<T>`	`vector`	Array dinamico.
`deque<T>`	`deque`	Array dinamico con inserimento anche all'inizio.
`set<T>`	`set`	Insieme dinamico di elementi.
`map<K, V>`	`map`	Dizionario dinamico che associa dei valori a delle chiavi.
`priority_queue<T>`	`queue`	Insieme che restituisce velocemente il minimo elemento.
`less<T>`	`functional`	Classe callable che confronta come un $<$.
`greater<T>`	`functional`	Classe callable che confronta come un $>$.

Algoritmi interessanti (in `algorithm`) sono:

`sort(v.begin(), v.end())`	Ordina in maniera crescente il vettore `v`.
`lower_bound(v.begin(), v.end(), k)`	Restituisce un iteratore al primo elemento non minore di `k`.
`min_element(v.begin(), v.end())`	Restituisce un iteratore all'elemento minimo.
`max_element(v.begin(), v.end())`	Restituisce un iteratore all'elemento massimo.

Ulteriori parametri opzionali sono disponibili per gli oggetti e algoritmi suindicati. Per esempio, in una `priority_queue` si può specificare l'ordinamento con una classe callable, quelle già fornite o una nuova del tipo:

```
class confronta {
  bool operator() (int x, int y) {
    return x < y;
  }
}
```